National University of Singapore
School of Computing
CS1010S: Programming Methodology
Semester I, 2018/2019

Recitation 9
Object-Oriented Programming

Problems

1. Write a `Food` class
   - Input properties is the `name`, `nutrition` value, and `good_until` time.
   - Additional property is the `age` of the food, initially 0.
   - Methods are:
     - `sit_there` - takes an amount of time, and increases the age of the food by the amount.
     - `eat` - return the nutrition if the food is still good; 0 otherwise.

2. Write an `AgedFood` class
   - Input property is the same as the `Food` class, with an additional property, which is the `good_after` time.
   - Should inherit from the `Food` class.
   - Methods are:
     - `sniff` - returns `True` if it has aged enough to be good, `False` otherwise.
     - `eat` - returns 0 if the food is not good yet; otherwise behaves like normal food.

3. Write a `VendingMachine` class
   - Input property is the same as the `Food` class.
   - Additional property is `age` of the `VendingMachine`, initially 0.
   - Methods are:
     - `sit_there` - takes an amount of time, and increases the age of the vending-machine by *half* that amount (it's refridgerated!).
     - `sell_food` - returns a new food instance with the appropriate name, nutrition and good_until.

4. Write `mapn`, which allows an arbitrary number of input lists[1], for example:

```
mapn(lambda x,y,z: (z, x+y),
  ((1, 2, 3), (4, 5, 6), ('first', 'second', 'third')))

#Output:        (('first', 5), ('second', 7), ('third', 9))
```

   The function definition should look like this:

```
def mapn(fn, lsts)
# fn is the function that you would apply to the lsts
# lsts is the list of lists that would given as input to the function fn
```

   You may use the regular `map` in your implementation.

5. **Homework:** How would you implement the vending machine so that it can sell both Food and AgedFood (and possibly other things too?).

[1] It turns out that the regular `map` is pretty similar to the `mapn` you will write here!